CS5340: Uncertainty Modeling in AI

Coding Assignment 1

Deadline: March 2, 2020 [2359 hours]

Problem 1. (Sum-Product Algorithm)

Your goal is to implement the sum-product algorithm for inference in directed trees with nodes representing discrete random variables.

Setup Description

The `main.py` script accepts a list of json files and outputs the number of marginals that were computed correctly by your program.

Example usage: `python main.py problem1.json`.

The `main` script first constructs a `tree.Tree` object based on the description in the JSON file and then calls the `inference.sum_product` function to compute the marginals. Finally, it compares the marginals computed by the `inference.sum_product` function to the correct marginals present in the JSON file.

Your task is to **only** modify the functions in `inference.py`. Your code will only be tested on tree-like Bayesian networks (no poly-trees). Refer to the comments in the code for details.

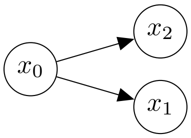

Figure 1: An example tree.

Structure of the JSON file

Let's consider the tree shown in Fig. 1 with the corresponding probability tables shown in Fig. 2. The following json file represents this tree.

The `nodes` key holds the number of nodes in the tree and their corresponding dimension where dimension implies the number of categories in the support of the random variable. For example, a dimension of 4 would mean that the random variable can take one of four possible values from $\{0, 1, 2, 3\}$. In the case of Fig. 1, nodes 0 & 1 represent binary random variables and node 2 represents a ternary random variable.

The `factors` key holds all the pair factors (or conditional distributions of the form $p(x|y)$); e.g., the `0` key holds the two factors between (0, 1) and (0, 2) represented by the probability tables in Fig. 2 (b) and (c) respectively.

The `observations` key holds the observations, if any.

The `p_0` key holds $p(x_0)$ (Fig. 2 (a)).

The `marginals` key holds the correct marginals obtained after running the sum-product algorithm. The marginals generated by your implementation should match these values.

$\bar{x}_0$	$p(x_0 = \bar{x}_0)$
0	0.1
1	0.9

(a) $p(x_0)$

$\bar{x}_0$	$p(x_1 = 0 \vert x_0 = \bar{x}_0)$	$p(x_1 = 1 \vert x_0 = \bar{x}_0)$
0	0.3	0.7
1	0.2	0.8

(b) $p(x_1|x_0)$

$\bar{x}_0$	$p(x_2 = 0 \vert x_0 = \bar{x}_0)$	$p(x_2 = 1 \vert x_0 = \bar{x}_0)$	$p(x_2 = 2 \vert x_0 = \bar{x}_0)$
0	0.2	0.3	0.5
1	0.6	0.1	0.3

(c) $p(x_2|x_0)$

Figure 2: Probability tables.

```
{"nodes":
    {"0": {"dim": 2},
     "1": {"dim": 2},
     "2": {"dim": 3}},
 "factors":
    {"0": {"1": [[0.3, 0.7],
                 [0.2, 0.8]],
           "2": [[0.2, 0.3, 0.5],
                 [0.6, 0.1, 0.3]]}
    },
 "observations": {},
 "p_0": [[0.1, 0.9]],
 "marginals":
    {"1": [[0.21, 0.79]],
     "2": [[0.56, 0.12, 0.32]],
     "0": [[0.1, 0.9]]
    }
}
```

The following json file contains the case when the variable x_2 has been observed to be 2.

```
{"nodes":
    {"0": {"dim": 2},
     "1": {"dim": 2},
     "2": {"dim": 3}},
 "factors":
    {"0": {"1": [[0.3, 0.7],
                 [0.2, 0.8]],
           "2": [[0.2, 0.3, 0.5],
                 [0.6, 0.1, 0.3]]}
    },
"observations": {"2": 2},
 "p_0": [[0.1, 0.9]],
 "marginals":
    {"1": [[0.215625, 0.784375]],
     "2": [[0.0, 0.0, 1.0]],
     "0": [[0.15625, 0.84375]]
    }
}
```

Some example JSON files have been included with the code for you to test your implementation. You can follow the structure of the JSON, as described above, and construct more problem files; the correct marginals can be obtained using a tool such as **Belief and Decision Networks**[1].

Deliverables

You are only required to modify `inference.py` and submit this file renamed to `YourMatricNumber.py`. For example, if your matric number is A0194057B, then the file should be named `A0194057B.py`. Upload this file to the `Assignment` folder on LumiNUS.

[1] http://www.aispace.org/downloads.shtml